Auto-tuning Multigrid with PetaBricks

Cy Chan

Joint Work with:

Jason Ansel
Yee Lok Wong
Saman Amarasinghe
Alan Edelman

Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Algorithmic Choice in Sorting

- Mergesort (N-way)
- Insertionsort
- Radixsort
- Quicksort
Algorithmic Choice in Sorting

Mergesort (N-way)  Insertionsort

N=2  @15

Radixsort  Quicksort

STL Algorithm
Algorithmic Choice in Sorting

Mergesort (N-way)

Insertionsort

Radixsort

Quicksort

Optimized For:
Xeon (1 core)
Algorithmic Choice in Sorting

Mergesort (N-way)

Insertionsort

Optimized For:
Xeon (1 core)
Xeon (8 cores)

Radixsort

Quicksort
Algorithmic Choice in Sorting

- **Mergesort (N-way)**: Optimized for Xeon (1 core)
  - N=2, 4, 8, 16
  - Times: @75, @1461, @2400

- **Insertionsort**
  - N=2
  - Times: @75

- **Radixsort**
  - N=4
  - Times: @98

- **Quicksort**
  - N=2
  - Times: @1420, @600

- **Optimized For**:
  - Xeon (1 core)
  - Xeon (8 cores)
  - Niagra (8 cores)
Variable Accuracy Algorithms

• Lots of algorithms where the accuracy of output can be tuned:
  – Iterative algorithms (e.g. solvers, optimization)
  – Signal processing (e.g. images, sound)
  – Approximation algorithms

• Can trade accuracy for speed

• All user wants: Solve to a certain accuracy as fast as possible using whatever algorithms necessary!
The PetaBricks Language

- **General purpose** language and auto-tuner
- Support for algorithmic choices and variable accuracy built into the language
- Specify multiple algorithms and accuracy levels
- Auto-tune parameters (e.g. number of iterations) to produce programs of different accuracy
- Multigrid is a prime target:
  - Iterative linear solver algorithm
  - Lots of choices!
Outline

• Auto-tuning with PetaBricks
• Tuning the Multigrid V-Cycle
• Extension to Auto-tuning Full Multigrid Cycles
• Performance Results
transform Sort
from A[n]
to B[n]
{
// Recursive case, merge sort
to(B b)
from(A a) {
    (a1, a2) = Split(a);
    b1 = Sort(a1);
    b2 = Sort(a2);
    b = Merge(b1, b2);
}

OR

// Base case, insertion sort
to(B b)
from(A a) {
    b = InsertionSort(a);
}
}
Modeling Costs

- Algorithmic Complexity
- Compiler Complexity
- Memory System Complexity
- Processor Complexity

All impact performance

- No simultaneous model for all of these!
- Solution: Use learning!
PetaBricks Work Flow

PetaBricks Source

PetaBricks Compiler

Tunable Executable

Configuration File

Static Executable
A Very Brief Multigrid Intro

- Used to iteratively solve PDEs over a gridded domain
- **Relaxations** update points using neighboring values (stencil computations)
- **Restrictions** and **Interpolations** compute new grid with coarser or finer discretization

![Diagram showing the multigrid process]

- Relax on current grid
- Restrict to coarser grid
- Interpolate to finer grid
Multigrid Cycles

Standard Approaches

V-Cycle

How coarse do we go?

W-Cycle

How many iterations?

Relaxation operator?

Full MG V-Cycle

Standard Approaches
Multigrid Cycles

• Generalize the idea of what a multigrid cycle can look like
• Example:

  • Goal: Auto-tune cycle shape for specific usage
Algorithmic Choice in Multigrid

- Need framework to make fair comparisons
- Perspective of a specific grid resolution
- How to get from A to B?

```
A B
Direct
A B
Iterative
```

```
A
Restrict
?
Interpolate
B
Recursive
```
Algorithmic Choice in Multigrid

• Tuning cycle shape!
  – Examples of recursive options:

  ![Standard V-cycle diagram](image)

Standard V-cycle
Algorithmic Choice in Multigrid

• Tuning cycle shape!
  – Examples of recursive options:

  Take a shortcut at a coarser resolution
Algorithmic Choice in Multigrid

• Tuning cycle shape!
  – Examples of recursive options:

A

Iterating with shortcuts

B
Algorithmic Choice in Multigrid

• Tuning cycle shape!
  – Once we pick a recursive option, how many times do we iterate?

Higher Accuracy

A B C D

• Number of iterations depends on what accuracy we want at the current grid resolution!
Comparing Cycle Shapes

• Different convergence AND execution rates
• Need a way to make fair comparisons
• Measure accuracy: reduction of RMS error
  – Example: A cycle has accuracy level $10^3$ if the RMS error of guess is reduced by a $10^3$ factor
  – Must train on representative data
  – Imperfect metric: ignores error frequency
• Use accuracy AND time to make comparisons between cycle shapes
Optimal Subproblems

- Plot all cycle shapes for a given grid resolution:

- Idea: Maintain a family of optimal algorithms for each grid resolution

Keep only the optimal ones!

- Idea: Maintain a **family** of optimal algorithms for each grid resolution
The Discrete Solution

• Problem: Too many optimal cycle shapes to remember

• Solution: Remember the fastest algorithms for a discrete set of accuracies
Use Dynamic Programming to Manage Auto-tuning Search

• Only search cycle shapes that utilize optimized sub-cycles in recursive calls
• Build optimized algorithms from the bottom up

• Allow shortcuts to stop recursion early
• Allow multiple iterations of sub-cycles to explore time vs. accuracy space
Auto-tuning the V-cycle

\[
\text{transform Multigrid}_k \\
\text{from } X[n,n], B[n,n] \\
\text{to } Y[n,n] \\
\{
// Base case \\
// Direct solve \\
\}
\]

\[
// Base case \\
// Iterative solve at current resolution \\
\]

\[
// Recursive case \\
// For some number of iterations \\
// Relax \\
// Compute residual and restrict \\
// Call Multigrid, for some i \\
// Interpolate and correct \\
// Relax \\
\}
\]

- **Algorithmic choice**
  - Shortcut base cases
  - Recursively call some optimized sub-cycle

- **Iterations and recursive accuracy** let us explore accuracy versus performance space

- **Only remember “best” versions**
Variable Accuracy Keywords

- **accuracy_variable** – tunable variable
- **accuracy_metric** – returns accuracy of output
- **accuracy_bins** – set of discrete accuracy bins
- **generator** – creates random inputs for accuracy measurement

```plaintext
transform Multigrid_k
from X[n,n], B[n,n]
to Y[n,n]
accuracy_variable numIterations
accuracy_metric Poisson2D_metric
accuracy_bins 1e1 1e3 1e5 1e7
generator Poisson2D_Generator
{
...
```
Training the Discrete Solution

Resolution $i$

Accuracy 1
- Multigrid Algorithm

Accuracy 2
- Multigrid Algorithm

Accuracy 3
- Multigrid Algorithm

Accuracy 4
- Multigrid Algorithm

Resolution $i+1$

Multigrid Algorithm

Optimized

Training
Training the Discrete Solution

Resolution $i$

<table>
<thead>
<tr>
<th>Accuracy 1</th>
<th>Accuracy 2</th>
<th>Accuracy 3</th>
<th>Accuracy 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multigrid Algorithm</td>
<td>Multigrid Algorithm</td>
<td>Multigrid Algorithm</td>
<td>Multigrid Algorithm</td>
</tr>
</tbody>
</table>

Resolution $i+1$

<table>
<thead>
<tr>
<th>Accuracy 1</th>
<th>Accuracy 2</th>
<th>Accuracy 3</th>
<th>Accuracy 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multigrid Algorithm</td>
<td>Multigrid Algorithm</td>
<td>Multigrid Algorithm</td>
<td>Multigrid Algorithm</td>
</tr>
</tbody>
</table>

Optimized

Optimized
Training the Discrete Solution

Accuracy 1  Accuracy 2  Accuracy 3  Accuracy 4

Multigrid Algorithm  Multigrid Algorithm  Multigrid Algorithm  Multigrid Algorithm

Multigrid Algorithm  Multigrid Algorithm  Multigrid Algorithm  Multigrid Algorithm

Multigrid Algorithm  Multigrid Algorithm  Multigrid Algorithm  Multigrid Algorithm

Multigrid Algorithm  Multigrid Algorithm  Multigrid Algorithm  Multigrid Algorithm

Finer

Coarser

Tuning order

Possible choice
(Shortcuts not shown)
Example: Auto-tuned 2D Poisson’s Equation Solver

4096 2048 1024 512 256 128 64 32

Accy. 10  Accy. 10^3  Accy. 10^7

Finer → Coarser
Auto-tuned Cycles for 2D Poisson Solver

Cycle shapes for accuracy levels a) $10$, b) $10^3$, c) $10^5$, d) $10^7$

Optimized substructures visible in cycle shapes
Extension to Full Multigrid

• Build auto-tuned Full Multigrid cycles out of auto-tuned V-cycles

• Two phases:
  – Estimation phase: Restrict and recursively call auto-tuned Full Multigrid at coarser grid resolution
  – Solve phase: Interpolate and run auto-tuned V-cycle at current grid resolution

• Choose accuracy level of each phase independently

• Use dynamic programming
Auto-tuned Full Multigrid Cycles for 2D Poisson Solver

Cycle shapes for accuracy levels a) 10, b) $10^3$, c) $10^5$, d) $10^7$
Benchmark Application: Solving 2D Poisson’s Equation

• Solve 2D Poisson’s Equation on random data (uniform over \([-2^{32}, 2^{32}]\)) for problems of size \(2^n\) for \(n = 2, 3, \ldots, 12\)

• Reference Algorithms (also in PetaBricks):
  – Reference Multigrid – Iterate using V-cycle until accuracy target is reached
  – Reference Full Multigrid – Estimate using a standard Full Multigrid iteration, then iterate using V-cycle until accuracy target is reached
Performance Testbed

• Shared memory machines
  – Intel Harpertown – Two quad-core 3.2 GHz Xeons
  – AMD Barcelona – Two quad-core 2.4 GHz Opterons
  – Sun Niagara – One quad-core 1.2 GHz T1

• PetaBricks compiler still under development
  – Some low-level optimizations not yet supported (no explicit pre-fetching or SIMD vectorization)
  – Focus on tuning and comparing cycle shapes
Impact of Auto-tuning Intel Harpertown (2 Sockets, 8 Cores)

The graph shows the relative time (ratio) for different configurations of the Intel Harpertown processor as a function of problem size. The configurations include:

- Reference V
- Reference Full MG
- Autotuned V
- Autotuned Full MG

The x-axis represents the problem size, while the y-axis represents the relative time (ratio) compared to a baseline. The graph illustrates the performance improvements achieved through auto-tuning compared to the reference configurations.
Impact of Auto-tuning
AMD Barcelona (2 Sockets, 8 Cores)
Impact of Auto-tuning
Sun Niagara (4 Cores, 32 Threads)
Tuned Cycles Across Architectures

Tuned cycles to achieve accuracy $10^5$ at resolution $2^{11}$

i) Intel Harpertown  ii) AMD Barcelona  iii) Sun Niagara
Selected Related Work

• Auto-tuning Software:
  – FFTW – Fast Fourier Transform
  – ATLAS, FLAME – Linear Algebra
  – SPARSITY, OSKI – Sparse Matrices
  – STAPL – Template Framework Library
  – SPL – Digital Signal Processing

• Tuning Multigrid:
  – SuperSolvers – Composite Linear Solver
    – Cache-Aware Multigrid
  – Thekale, Gradl, Klamroth, Rude (2009) – Optimizing Interations of V-Cycles in Full Multigrid
Future Work

• Add support for auto-tuning other aspects of multigrid
  – Tuning of relaxation, interpolation, and restriction operators
  – Low-level optimizations: explicit prefetch and vectorization

• Add support for tuning data movement in AMR
  – Parameterize tuned subproblems by data location in addition to size and accuracy
  – Try different data layouts during recursion
General PetaBricks
Future Work

• Dynamic choices during execution

• Support for other parallel architectures
  – Distributed memory machines
  – Heterogeneous clusters (e.g. CPU + GPGPU)

• Sparse Matrix support
  – Auto-tune sparse matrix storage format
    – e.g. CSR, CSC, COO, ELLPACK
    – register block sizes, cache block sizes
Conclusion

• Auto-tuning with PetaBricks
  – Algorithmic choice
  – Variable accuracy

• Auto-tuning Multigrid Cycles
  – Construct more efficient multigrid solvers
  – Use dynamic programming
  – Speedup shown over reference algorithms
Thanks!

http://projects.csail.mit.edu/petabricks/